**Q1. Is it permissible to use several import statements to import the same module? What would the goal be? Can you think of a situation where it would be beneficial?**

you can import modules as many times as you want in one Python program, no matter what module. Every subsequent import after the first accesses the cached module instead of re-evaluating it.

The module is only loaded the first time the import statement is executed and there is no performance loss by importing it again.

You can import multiple items from the same module at once by listing them, separated by commas

**Q2. What are some of a module's characteristics? (Name at least one.)**

A module is a Python object with arbitrarily named attributes that you can bind and reference. Simply, a module is a file consisting of Python code. A module can define functions, classes and variables. A module can also include runnable code

**Q3. Circular importing, such as when two modules import each other, can lead to dependencies and bugs that aren't visible. How can you go about creating a program that avoids mutual importing?**

In general, circular imports are the result of bad designs. A deeper analysis of the program could have concluded that the dependency isn't actually required, or that the depended functionality can be moved to different modules that wouldn't contain the circular reference.

module 1 & 2

def function1():

function2()

def function2():

print('Hello, World!')

function3()

def function3():

print('Goodbye, World!')

function1()

A simple solution is that sometimes both modules can just be merged into a single, larger module. The resulting code from our example above would look something like this:

Q4. **Why is \_ \_all\_ \_ in Python?**

Python \_\_all\_\_ is a variable that can be set in the \_\_init\_\_.py file of a package. The \_\_all\_\_ variable is a list of strings that defines those symbols that are imported when a program runs.

Before understanding how the Python \_\_all\_\_ variable works, let us understand what is Python module and how we can import it.

**Q5. In what situation is it useful to refer to the \_ \_name\_ \_ attribute or the string '\_ \_main\_ \_'?**

The \_\_name\_\_ variable (two underscores before and after) is a special Python variable. It gets its value depending on how we execute the containing script. Sometimes you write a script with functions that might be useful in other scripts as well. In Python, you can import that script as a module in another script

**Q6. What are some of the benefits of attaching a program counter to the RPN interpreter application, which interprets an RPN script line by line?**

Reverse Polish notation (also known as postfix notation) is one of multiple notations for representing mathematical expressions. Most of us are familiar with infix notation, but there are also the less popular prefix and postfix notations.

**Q7. What are the minimum expressions or statements (or both) that you'd need to render a basic programming language like RPN primitive but complete— that is, capable of carrying out any computerised task theoretically possible?**

# Ambiguity in order of operations for infix notation

1 + 2 + 3

(1 + 2) + 3

1 + (2 + 3)

# No ambiguity in prefix and postfix (using postfix as an example)

# The structure tells us that 1 and 2 are added first, then 3

1 2 + 3 +